Square Root SAM

Simultaneous Localization and Mapping via Square Root Information Smoothing

Frank Dellaert
College of Computing
Georgia Institute of Technology
Technical Report No. GIT-GVU-05-11

April 2005

Abstract

Solving the SLAM problem is one way to enable a robot to explore, map, and navigate in a previously unknown environment. We investigate smoothing approaches as a viable alternative to extended Kalman filter-based solutions to the problem. In particular, we look at approaches that factorize either the associated information matrix or the measurement matrix into square root form. Such techniques have several significant advantages over the EKF: they are faster yet exact, they can be used in either batch or incremental mode, are better equipped to deal with non-linear process and measurement models, and yield the entire robot trajectory, at lower cost. In addition, in an indirect but dramatic way, column ordering heuristics automatically exploit the locality inherent in the geographic nature of the SLAM problem.

In this paper we present the theory underlying these methods, an interpretation of factorization in terms of the graphical model associated with the SLAM problem, and simulation results that underscore the potential of these methods for use in practice.

1 Introduction

The problem of simultaneous localization and mapping (SLAM) [1, 2, 3] has received considerable attention in mobile robotics as it is one way to enable a robot to explore, map, and navigate in previously unknown environments. The traditional approach in the literature is to phrase the problem as an extended Kalman filter (EKF), with the robot pose and static landmarks as the evolving filter state [4, 2, 5, 6]. It is well known that the computational requirements of the EKF become impractical fairly quickly, once the number of landmarks in the environment grows beyond a few hundred. As a result, many authors have looked at ways to reduce the computation associated with the EKF, using both approximating [7, 8, 9] and non-approximating [10] algorithms.

In this paper we propose that *square root information smoothing* (SRIS) is a fundamentally better approach to the problem of SLAM than the EKF, based on the realization that,

- in contrast to the extended Kalman filter covariance or information matrix, which *both* become fully dense over time [8, 9], the information matrix $\mathcal{I}$ associated with *smoothing* is and stays sparse;
- in typical mapping scenarios this matrix $\mathcal{I}$ or, alternatively, the measurement matrix A, are much more compact representations of the map covariance structure
- $\mathcal{I}$ or A, both sparse, can be factorized efficiently using sparse Cholesky or QR factorization, respectively, yielding a square root information matrix R that immediately yields the optimal robot trajectory and map;

Factoring the information matrix is known in the sequential estimation literature as square root information filtering (SRIF). The SRIF was developed in 1969 for use in JPL's Mariner 10 missions to Venus (as recounted by [11]), following the development of *covariance* square root filters earlier that decade. The use of square roots of either the covariance or information matrix results in more accurate and stable algorithms, and, quoting Maybeck [12] "a number of practitioners have argued, with considerable logic, that square root filters should *always* be adopted in preference to the standard Kalman filter recursion". Maybeck briefly discusses the SRIF in a chapter on square root filtering, and it and other square root type algorithms are the subject of a book by Bierman [11]. However, as far as this can be judged by the small number of references in the literature, the SRIF and the square root information *smoother* (SRIS) are not often used.

In this paper we investigate the use of factorizing either the information matrix $\mathcal{I}$ or the measurement matrix A into square root form, as applied to the problem of simultaneous *smoothing and mapping* (SAM). Because they are based on matrix square roots, we will refer to this family of approaches as *square root SAM*, or $\sqrt{\text{SAM}}$ for short. They have several significant advantages over the EKF:

- They are much faster than EKF-based SLAM
- They are exact, rather than approximate
- They can be used in either batch or incremental mode
- If desired, they yield the *entire* smoothed robot trajectory
- They are much better equipped to deal with non-linear process and measurement models than the EKF
- When using QR, they are more accurate and stable
- They *automatically* exploit locality in the the way that sub-map [7] or compressed filter [10] SLAM variants do

However, there is also a price to pay:

- Because we smooth the entire trajectory, computational complexity grows without bound over time, for both Cholesky and QR factorization strategies. In many typical mapping scenarios, however, the EKF information matrix will grow much faster.
- As with all information matrix approaches, it is expensive to recover the covariance matrix governing the unknowns.

We also present an interpretation of the resulting algorithms in terms of graphical models, following [13, 14, 8, 9]. Doing so yields considerable insight into the workings of otherwise opaque "black box" algorithms such as Cholesky or QR factorization. It exposes their kinship with recently developed inference methods for graphical models [15], such as the junction tree algorithm. In particular, in Section 2 we introduce the SLAM problem in terms of a directed graph or belief net. However, to understand factorization, the view of smoothing in terms of an *undirected* Markov random field (MRF), introduced in Section 5, is more appropriate. *By exploiting the SLAM-specific graph structure, we are able to immediately speed up factorization by a factor of 2.* We believe that even more efficient algorithms can be developed by viewing the problem as one of computation on a graph.

2 SLAM Background

SLAM refers to the problem of localizing a robot while simultaneously mapping its environment, illustrated in Figure 1. Below we assume familiarity with EKF-based approaches to SLAM [4, 2, 5, 6]. In this section we introduce the SLAM problem and the notation we use, but we do not re-derive the extended Kalman filter. Rather, in Section 3 we immediately take a smoothing approach, in which both the map and the robot trajectory are recovered.

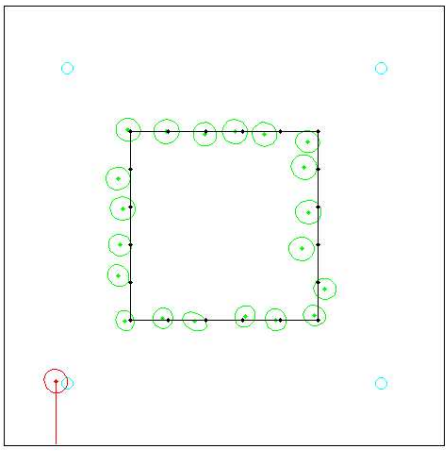


Figure 1: A synthetic environment with 20 landmarks in which we simulated a robot taking 156 bearing and range measurements along a trajectory of 21 poses. Also shown are the mean and covariance matrices as estimated by an EKF. Note the effects of process and measurement noise.

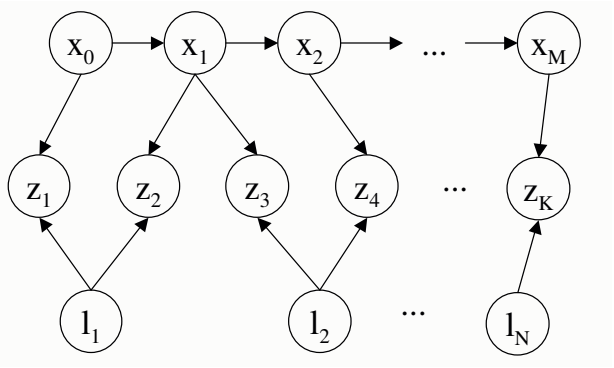


Figure 2: Bayesian belief network representation of the SLAM problem. The state x of the robot is governed by a Markov chain, on top, and the environment of the robot is represented at the bottom by a set of landmarks l. The measurements z, in the middle layer, are governed both by the state of the robot and the parameters of the landmark measured.

Following the trend set by FastSLAM and others [13, 14, 8], we formulate the problem by referring to a belief net representation. The model we adopt is shown in Figure 2. Here we denote the state of the robot at the i^{th} time step by x_i, with $i \in 0..M$, a landmark by l_j, with $j \in 1..N$, and a measurement by z_k, with $k \in 1..K$. The joint probability model corresponding to this network is

$$P(X, M, Z) = P(x_0) \prod_{i=1}^{M} P(x_i|x_{i-1}, u_i) \times \prod_{k=1}^{K} P(z_k|x_{i_k}, l_{j_k}) \quad (1)$$

where $P(x_0)$ is a prior on the initial state of the robot, $P(x_i|x_{i-1}, u_i)$ is the *motion model*, parameterized by a control input u_i, and $P(z|x, l)$ is the *landmark measurement model*. The above assumes a uniform prior over the landmarks l. Furthermore, it assumes that the data-association problem has been solved, i.e., that the indices i_k and j_k corresponding to each measurement z_k are known.

As is standard in the SLAM literature [4, 2, 5, 6], we assume Gaussian process and measurement models [12], defined by

$$x_i = f_i(x_{i-1}, u_i) + w_i \quad (2)$$

where $f_i(.)$ is a process model, and w_i is normally distributed zero-mean process noise with covariance matrix Λ_i, and

$$z_k = h_k(x_{i_k}, l_{j_k}) + v_k \quad (3)$$

where $h_k(.)$ is a measurement equation, and v_k is normally distributed zero-mean measurement noise with covariance Σ_k. The equations above model the robot's behavior in response to control input, and its sensors, respectively.

For the prior $P(x_0)$, *we will assume that x_0 is given* and hence it is treated as a constant below. This considerably simplifies the equations in the rest of this document. In addition, this is what is often done in practice: the origin of the coordinate system is arbitrary, and we can then just as well fix x_0 at the origin. The exposition below is easily adapted to the case where this assumption is invalid.

Below we also need the first-order linearized version of the process model (2), given by

$$x_i^0 + \delta x_i = f_i(x_{i-1}^0, u_i) + F_i^{i-1} \delta x_{i-1} + w_i \quad (4)$$

where F_i^{i-1} is the Jacobian of $f_i(.)$ at the linearization point x_{i-1}^0, defined by

$$F_i^{i-1} \triangleq \left. \frac{\partial f_i(x_{i-1}, u_i)}{\partial x_{i-1}} \right|_{x_{i-1}^0}$$

Note that u_i is given and appears as a constant above. The linearized measurement equations are obtained similarly,

$$z_k = h_k(x^0_{i_k}, l^0_{j_k}) + H^{i_k}_k \delta x_{i_k} + J^{j_k}_k \delta l_{j_k} + v_k \tag{5}$$

where $H^{i_k}_k$ and $J^{j_k}_k$ are respectively the Jacobians of $h_k(.)$ with respect to a change in $x_{i_k}$and l_{j_k}, and are evaluated at the linearization point $(x^0_{i_k}, l^0_{j_k})$:

$$H^{i_k}_k \triangleq \left.\frac{\partial h_k(x_{i_k}, l_{j_k})}{\partial x_{i_k}}\right|_{(x^0_{i_k}, l^0_{j_k})} \quad J^{j_k}_k \triangleq \left.\frac{\partial h_k(x_{i_k}, l_{j_k})}{\partial l_{j_k}}\right|_{(x^0_{i_k}, l^0_{j_k})}$$

3 Smoothing SLAM and Least Squares

We investigate smoothing rather than filtering, i.e., we are interested in recovering the maximum a posteriori (MAP) estimate for the *entire* trajectory $X \triangleq \{x_i\}$ and the map $L \triangleq \{l_j\}$, given the measurements $Z \triangleq \{z_k\}$ and control inputs $U \triangleq \{u_i\}$. Let us collect all unknowns in X and L in the vector $\theta \triangleq (X, L)$. Under the assumptions made above, we obtain the MAP estimate

$$\begin{aligned} \theta^* \triangleq \underset{\theta}{\text{argmax}} \;\; P(X, L|Z) &= \underset{\theta}{\text{argmax}} \;\; P(X, L, Z) \\ &= \underset{\theta}{\text{argmin}} \;\; -\log P(X, L, Z) \end{aligned}$$

by solving the following non-linear least-squares problem:

$$\sum_{i=1}^{M} \|x_i - f_i(x_{i-1}, u_i)\|^2_{\Lambda_i} + \sum_{k=1}^{K} \|z_k - h_k(x_{i_k}, l_{j_k})\|^2_{\Sigma_k} \tag{6}$$

Here $\|e\|^2_\Sigma \triangleq e^T \Sigma^{-1} e$ is defined as the squared Mahalanobis distance given a covariance matrix Σ.

In practice one always considers a linearized version of this problem. If the process models f_i and measurement equations h_k are non-linear and a good linearization point is not available, non-linear optimization methods such as Gauss-Newton iterations or Levenberg-Marquardt will solve a succession of linear approximations to (6) in order to approach the minimum [16]. This is similar to the extended Kalman filter approach to SLAM as pioneered by [17, 4, 18], but allows for iterating multiple times to convergence.

In what follows, we will assume that either a good linearization point is available or that we are working on one of these iterations. In either case, we have a *linear* least-squares

problem that needs to be solved efficiently. Using the linearized process and measurement models (4) and (5), respectively, we obtain

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{M} \|F_i^{i-1} x_{i-1} + G_i^i x_i - a_i\|_{\Lambda_i}^2 + \sum_{k=1}^{K} \|H_k^{i_k} x_{i_k} + J_k^{j_k} l_{j_k} - c_k\|_{\Sigma_k}^2 \tag{7}$$

where we define $a_i \triangleq x_i^0 - f_i(x_{i-1}^0, u_i)$ and $c_k \triangleq z_k - h_k(x_{i_k}^0, l_{j_k}^0)$. To avoid treating x_i in a special way we also introduce the matrix $G_i^i = -I_{d\times d}$, with d the dimension of the robot state, and we drop the δ notation as implied.

Below we assume, without loss of generality, that the covariance matrices Λ_i and Σ_k are all unity. Because of

$$\|e\|_\Sigma^2 \triangleq e^T \Sigma^{-1} e = (\Sigma^{-T/2} e)^T (\Sigma^{-T/2} e) = \left\|\Sigma^{-T/2} e\right\|_2^2$$

with $\Sigma^{-1/2}$ the matrix square root of Σ, we can always eliminate Λ_i from (7) by pre-multiplying F_i^{i-1}, G_i^i, and a_i in each term with $\Lambda_i^{-T/2}$. A similar story holds for the matrices Σ_k, where *for scalar measurements this simply means dividing each term by the measurement standard deviation.* Below we assume that this has been done and drop the Mahalanobis norm in favor of the regular 2-norm.

Finally, after collecting the Jacobian matrices into a matrix A, and the vectors a_i and c_k into a right-hand side (RHS) vector b, we obtain the following standard least-squares problem,

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \ \|A\theta - b\|_2^2 \tag{8}$$

which is our starting point below. A can grow to be very large, but is quite sparse, as shown in Figure 3. If d_x, d_l, and d_z are the dimensions of the state, landmarks, and measurements, A's size is $(Nd_x + Kd_z) \times (Nd_x + Md_l)$. In addition, A has a typical block structure, e.g., with $M = 3$, $N = 2$, and $K = 4$:

$$A = \begin{bmatrix} G_1^1 & & & & & \\ F_2^1 & G_2^2 & & & & \\ & F_3^2 & G_3^3 & & & \\ H_1^1 & & & J_1^1 & & \\ H_2^1 & & & & J_2^2 & \\ & H_3^2 & & J_3^1 & & \\ & & H_4^3 & & J_4^2 & \end{bmatrix}$$

Above the top half describes the robot motion, and the bottom half the measurements. A mixture of landmarks and/or measurements of different types (and dimensions) is easily accommodated. Note that the non-zero blocks pattern of the measurement part is also the adjacency matrix for the measurement part of a belief net like the one in Figure 2.

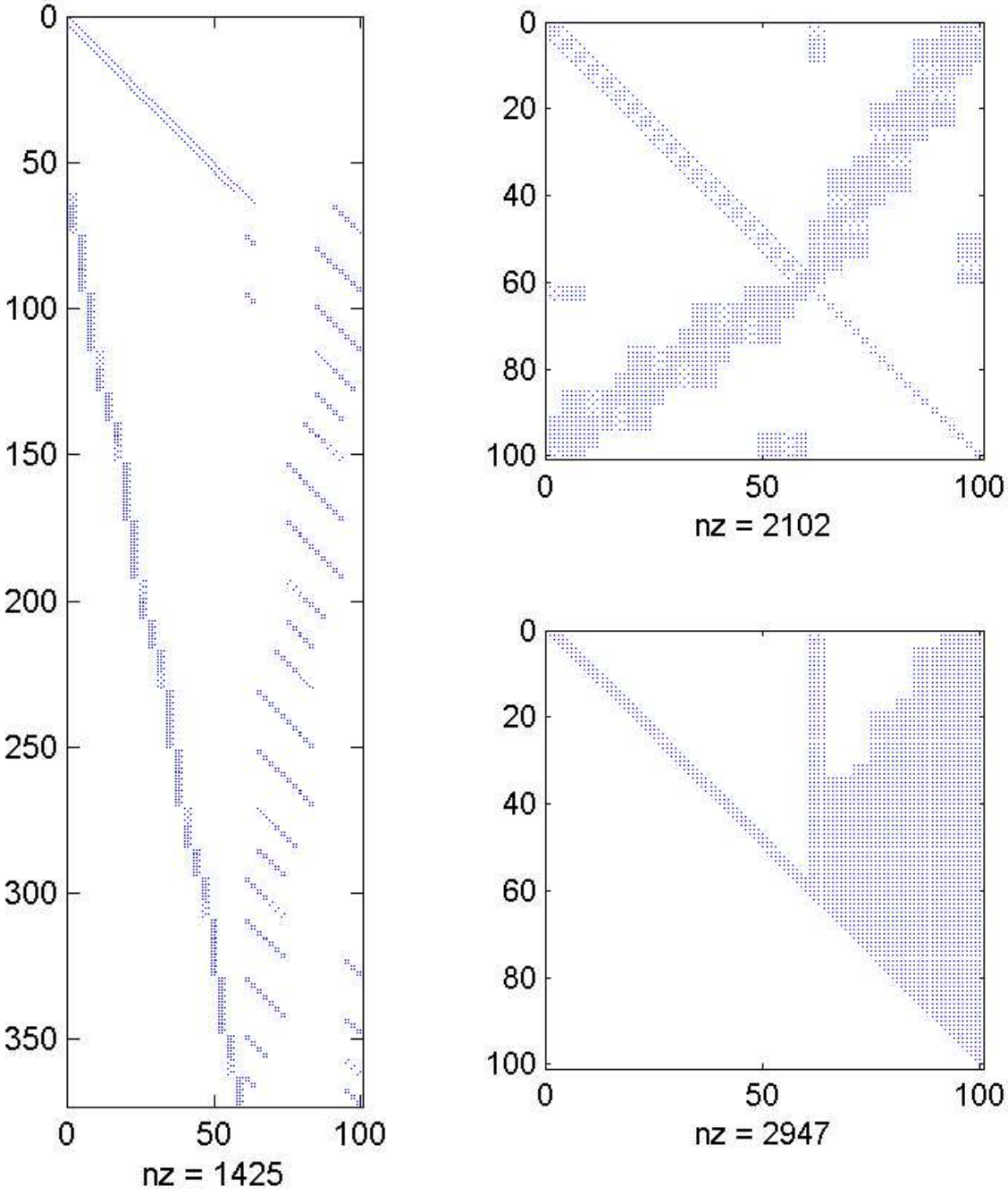


Figure 3: Measurement matrix A associated with the problem in Figure 1, along with the information matrix $\mathcal{I} \triangleq A^T A$, and its Cholesky triangle R. Here the state is 3D and both landmarks and measurements are 2D, hence the size of A is $(20\times3+156\times2)\times(20\times3+20\times2) = 372 \times 100$.

4 Cholesky and QR Factorization

In this section we briefly review Cholesky and QR factorization and their application to the full rank linear least-squares (LS) problem in (8). The exposition closely follows [19], which can be consulted for a more in-depth treatment.

For a full-rank $m \times n$ matrix A, with $m \geq n$, the unique LS solution to (8) can be found by solving the *normal equations*:

$$A^T A\theta^* = A^T b \tag{9}$$

This is normally done by Cholesky factorization of the *information matrix* $\mathcal{I}$, defined and factorized as follows:

$$\mathcal{I} \triangleq A^T A = R^T R \tag{10}$$

The *Cholesky triangle* R is an upper-triangular $n \times n$ matrix[1] and is computed using *Cholesky factorization*, a variant of LU factorization for symmetric positive definite matrices. It runs in $n^3/3$ flops. After this, θ^* can be found by solving

$$\text{first } R^T y = A^T b \text{ and then } R\theta^* = y$$

by back-substitution. The entire algorithm, including computing half of the symmetric $A^T A$, requires $(m + n/3)n^2$ flops.

For the example of Figure 1, both $\mathcal{I}$ and its Cholesky triangle R are shown alongside A in Figure 3. Note the very typical block structure of $\mathcal{I}$ when the columns of A are ordered in the canonical way, i.e., trajectory X first and then map L:

$$\mathcal{I} = \begin{bmatrix} A_X^T A_X & \mathcal{I}_{XL} \\ \mathcal{I}_{XL}^T & A_L^T A_L \end{bmatrix}$$

Here $\mathcal{I}_{XL} \triangleq A_X^T A_L$ encodes the correlation between robot states X and map L, and the diagonal blocks are band-limited.

An alternative to Cholesky factorization that is both more accurate and numerically stable is to proceed via QR-factorization *without* computing the information matrix $\mathcal{I}$. Instead, we compute the QR-factorization of A itself along with its corresponding RHS:

$$Q^T A = \begin{bmatrix} R \\ 0 \end{bmatrix} \quad Q^T b = \begin{bmatrix} d \\ e \end{bmatrix}$$

Here Q is an $m \times m$ orthogonal matrix, and R is the upper-triangular Cholesky triangle. The preferred method for factorizing a dense matrix A is to compute R column by column, proceeding from left to right. For each column j, all non-zero elements below the diagonal

[1] Some treatments, including [19], define the Cholesky triangle as the lower-triangular matrix $G = R^T$, but the other convention is more convenient here.

are zeroed out by multiplying A on the left with a *Householder reflection matrix* H_j. After n iterations A is completely factorized:

$$H_n..H_2H_1A = Q^TA = \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{11}$$

The orthogonal matrix Q is not usually formed: instead, the transformed RHS Q^Tb is computed by appending b as an extra column to A. Because the Q factor is orthogonal, we have:

$$\|A\theta - b\|_2^2 = \left\|Q^TA\theta - Q^Tb\right\|_2^2 = \|R\theta - d\|_2^2 + \|e\|_2^2$$

Clearly, $\|e\|_2^2$ will be the least-squares residual, and the LS solution θ^* can be obtained by solving the square system

$$R\theta = d \tag{12}$$

via back-substitution. The cost of QR LS is dominated by the cost of the Householder reflections, which is $2(m - n/3)n^2$.

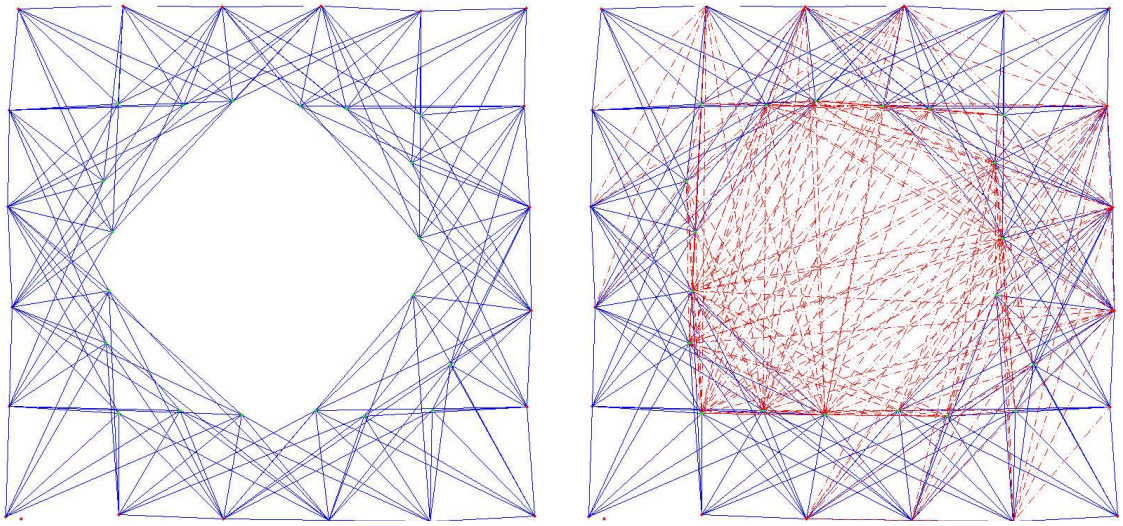
Comparing QR with Cholesky factorization, we see that both algorithms require $O(mn^2)$ operations when $m \gg n$, but that QR-factorization is a factor of 2 slower. However, this is only for *dense* matrices: if A is *sparse*, as is the case in the SLAM problem, QR factorization becomes quite competitive.

5 A Graphical Model Perspective

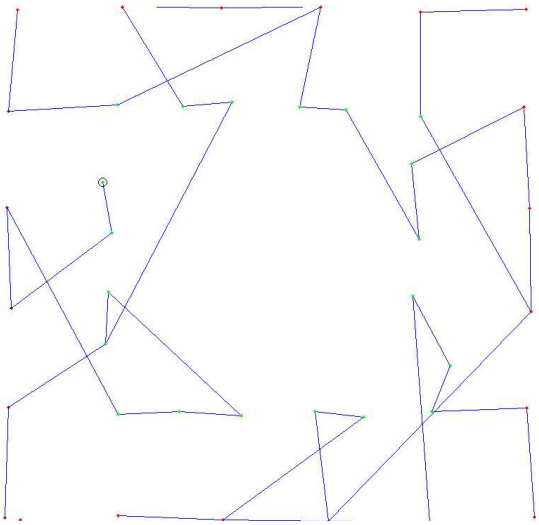
Cholesky or QR factorization are most often used as "black box" algorithms, but in fact they are surprisingly similar to much more recently developed methods for inference in graphical models [15]. Taking a graphical model view on SLAM exposes its sparse structure in full, and shows how sparse factorization methods in this context operate on a graph.

When examining the correlation structure of the problem it is better to eliminate Z and consider the *undirected* graph that encodes the correlations between the unknowns θ only. In [8, 9] this view is taken to expose the correlation structure inherent in the *filtering version* of SLAM. It is shown there that inevitably, when marginalizing out the past trajectory $X_{1:m-1}$, the information matrix becomes completely dense. Hence, the emphasis in these approaches is to selectively remove links to reduce the computational cost of the filter, with great success.

In contrast, in this paper we consider the graph associated with the *smoothing* information matrix $\mathcal{I} \triangleq A^TA$, which does *not* become dense, as past states are never marginalized out. In particular, the objective function in Equation 6 corresponds to a pairwise Markov random field (MRF) [20, 21] through the Hammersley-Clifford theorem [20]. The nodes in the MRF correspond to the robot states and the landmarks, and links represent either odometry or landmark measurements. The resulting bipartite graph corresponding to the example of Figure 1 is shown in Figure 4.



(a) (b)



(c)

Figure 4: a) The graph of the Markov random field of the associated SLAM problem from Figure 1. b) The triangulated graph: each edge corresponds to a non-zero in the Cholesky triangle R. c) The corresponding elimination tree showing how the state and landmarks estimates will be computed via back-substitution: the root is computed first - in this case a landmark near the top left - after which the computation progresses further down the tree.

Both QR and Cholesky factorization *eliminate* one variable at a time, starting with θ_1, corresponding in the leftmost column of either A or $\mathcal{I}$. The result of the elimination is that θ_1 is now expressed as a linear combination of all other unknowns $\theta_{j>1}$, with the coefficients residing in the corresponding row R_1 of R. In the process, however, new dependencies are introduced between all variables connected to θ_1, *which causes edges to be added to the graph*. The next variable is then treated in a similar way, until all variables have been eliminated. This is exactly the process of moralization and triangulation familiar from graphical model inference. The result of eliminating all variables is a chordal graph, shown for our example in Figure 4b.

The single most important factor to good performance is the order in which variables are eliminated. Different variable orderings can yield dramatically more or less *fill-in*, defined as the amount of edges added into the graph. As each edge added corresponds to a non-zero in the Cholesky triangle R, both the cost of computing R and back-substitution is heavily dependent on how much fill-in occurs. Unfortunately, finding an optimal ordering is NP-complete. Discovering algorithms that approximate the optimal ordering is an active research area in linear algebra. A popular method for medium-sized problems is *colamd* [22]. However, as we will show in Section 7, using domain knowledge can do even better.

A data structure that underlies many of these approximate column ordering algorithms is the *elimination tree*. It is defined as a depth-first spanning tree of the chordal graph after elimination, and is useful in illustrating the flow of computation during the back-substitution phase. The elimination tree corresponding to our example, for a good column ordering, is shown in Figure 4c. The root of the tree corresponds to the last variable θ_n to be eliminated, which is the first to be computed in back-substitution (Equation 12). Computation then proceeds down the tree, and while this is typically done in reverse column order, variables in disjoint subtrees may be computed in any order. In fact, if one is only interested in certain variables, there is no need to compute any of the subtrees that do not contain them. The elimination tree is also the basis for multifrontal QR methods [23], which we have also evaluated in our simulations below.

6 Square Root SAM

A batch-version of *square root information smoothing and mapping* is straightforward and a completely standard way of solving a large, sparse least-squares problem:

Algorithm 1 Batch $\sqrt{\text{SAM}}$

---

1. Build the measurement matrix A and the RHS b as explained in Section 3.
2. Find a good column ordering p, and reorder $A_p \overset{p}{\leftarrow} A$
3. Solve $\theta_p^* = \text{argmin}_{\ \theta} \|A_p\theta_p - b\|_2^2$ using either the Cholesky or QR factorization method from Section 4
4. Recover the optimal solution by $\theta \overset{r}{\leftarrow} \theta_p$, with $r = p^{-1}$

In tests we have obtained the best performance with sparse LDL factorization [24], a variant on Cholesky factorization that computes $\mathcal{I} = LDL^T$, with D a diagonal matrix and L a lower-triangular matrix with ones on the diagonal.

We have also experimented with finding better, SLAM specific column re-orderings. A simple idea is to use a standard method such as *colamd*, but have it work on the sparsity pattern of the blocks instead of passing it the original measurement matrix A. This amounts to working directly with the bipartite MRF graph from Section 5, making accidental zeros due to the linearization invisible. Surprisingly, as we will show, the symbolic factorization on this restricted graph yields *better* column orderings.

Incremental $\sqrt{\text{SAM}}$

In a robotic mapping context, an incremental version of the above algorithm is of interest. It is well known that factorizations can be updated incrementally. One possibility is to use a rank 1 Cholesky update, a standard algorithm that computes the factor R' corresponding to a $\mathcal{I}' = \mathcal{I} + aa^T$, where a^T is a new row of the measurement matrix A. However, these algorithms are typically implemented for dense matrices only, and it is imperative that we use a sparse storage scheme for optimal performance. While sparse Cholesky updates exist [25], they are relatively complicated to implement. A second possibility, easy to implement and suited for sparse matrices, is to use a series of Givens rotations (see [19]) to eliminate the non-zeros in the new measurement rows one by one.

A third possibility, which we have adopted for the simulations below, is to update the matrix $\mathcal{I}$ and simply use a full Cholesky (or LDL) factorization. QR factorization is more accurate and has better numerical properties, but a Cholesky or LDL factorization can be corrected with one linear update step to achieve the same accuracy, if required.

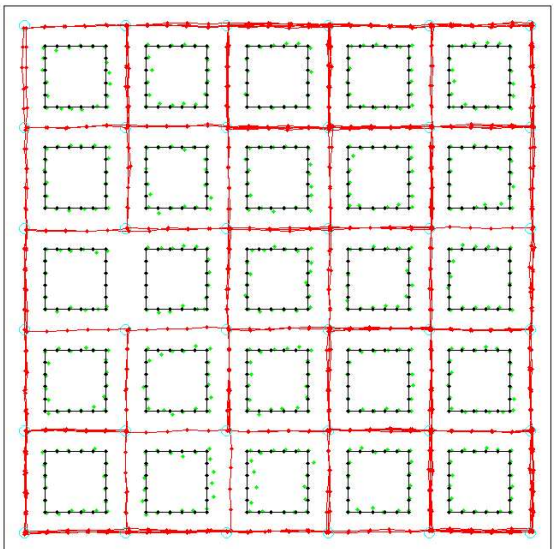


Figure 5: A synthetic environment with 500 landmarks along with a 1000-step random walk trajectory, corresponding to 14000 measurements taken.

Importantly, because the entire measurement history is implicit in $\mathcal{I}$, one does not need to factorize at every time-step. In principle, we can wait until the very end and then compute the entire trajectory and map. At any time during an experiment, however, the map and/or trajectory can be computed by a simple factorization and back-substitution, e.g., for visualization and/or path planning purposes.

7 Initial Findings and Simulation Results

7.1 Batch $\sqrt{\text{SAM}}$

We have experimented at length with different implementations of Cholesky, LDL, and QR factorization to establish which performed best. All simulations were done in MATLAB on a 2GHz. Pentium 4 workstation running Linux. Experiments were run in synthetic environments like the one shown in Figure 5, with 180 to 2000 landmarks, for trajectories of length 200, 500, and 1000. Each experiment was run 10 times for 5 different methods:

- *none*: no factorization performed
- *ldl*: Davis' sparse LDL factorization [24]
- *chol*: MATLAB built-in Cholesky factorization

M	N	none	ldl	chol	mfqr	qr
200	180	0.031	0.062	0.092	0.868	1.685
	500	0.034	0.062	0.094	1.19	1.256
	1280	0.036	0.068	0.102	1.502	1.21
	2000	0.037	0.07	0.104	1.543	1.329
500	180	0.055	0.176	0.247	2.785	11.92
	500	0.062	0.177	0.271	3.559	8.43
	1280	0.068	0.175	0.272	5.143	6.348
	2000	0.07	0.181	0.279	5.548	6.908
1000	180	0.104	0.401	0.523	10.297	42.986
	500	0.109	0.738	0.945	12.112	77.849
	1280	0.124	0.522	0.746	14.151	35.719
	2000	0.126	0.437	0.657	15.914	25.611

Figure 6: Averaged simulation results over 10 tries, in seconds, for environments with various number of landmarks N and simulations with trajectory lengths M. The methods are discussed in more detail in the text. The *none* method corresponds to doing no factorization and measures the overhead.

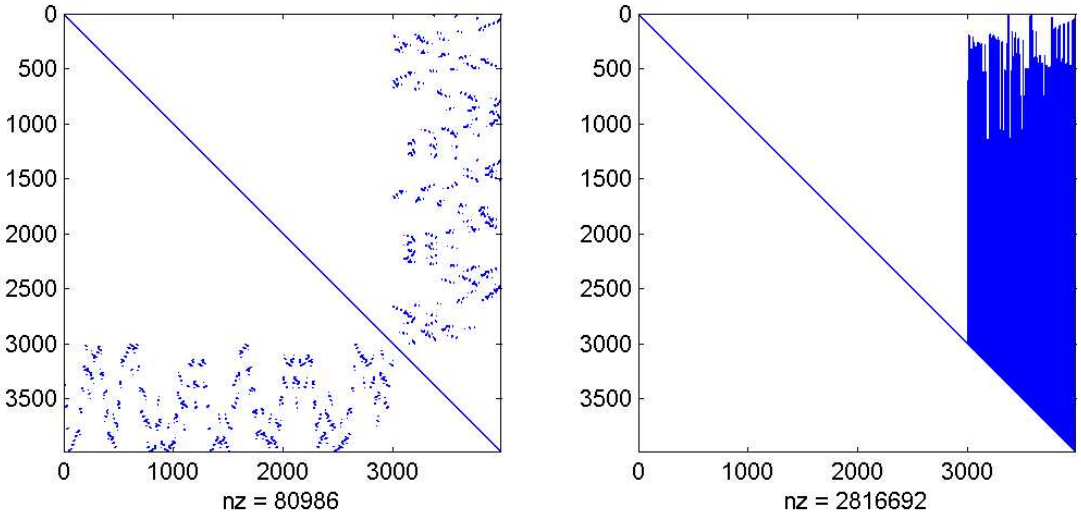


Figure 7: Original information matrix $\mathcal{I}$ and its Cholesky triangle. Note the dense fill-in on the right, linking the entire trajectory to all landmarks.

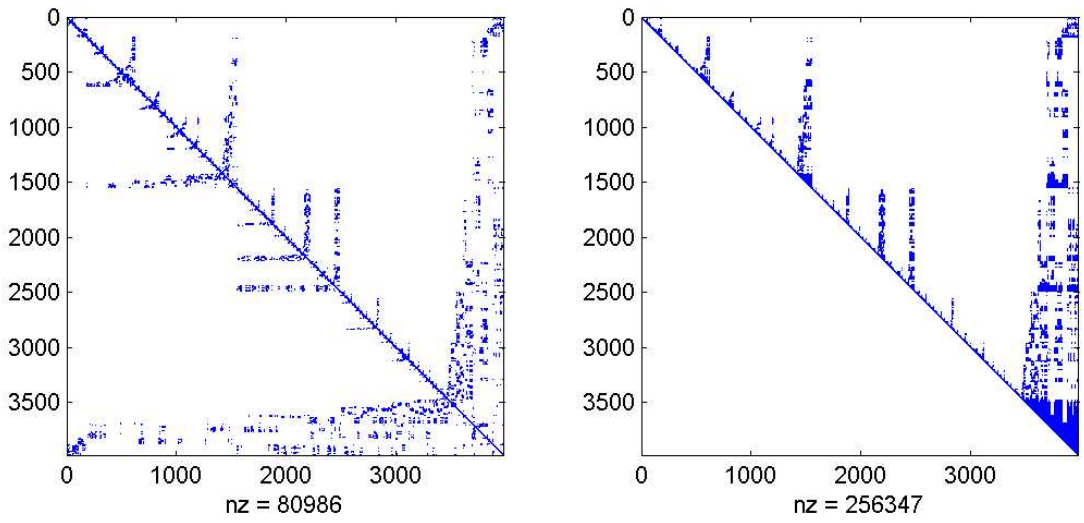


Figure 8: Information matrix $\mathcal{I}$ after reordering and its Cholesky triangle. Reordering of columns (unknowns) does not affect the sparseness of $\mathcal{I}$, but the number of non-zeroes in R has dropped from approximately 2.8 million to about 250 thousand.

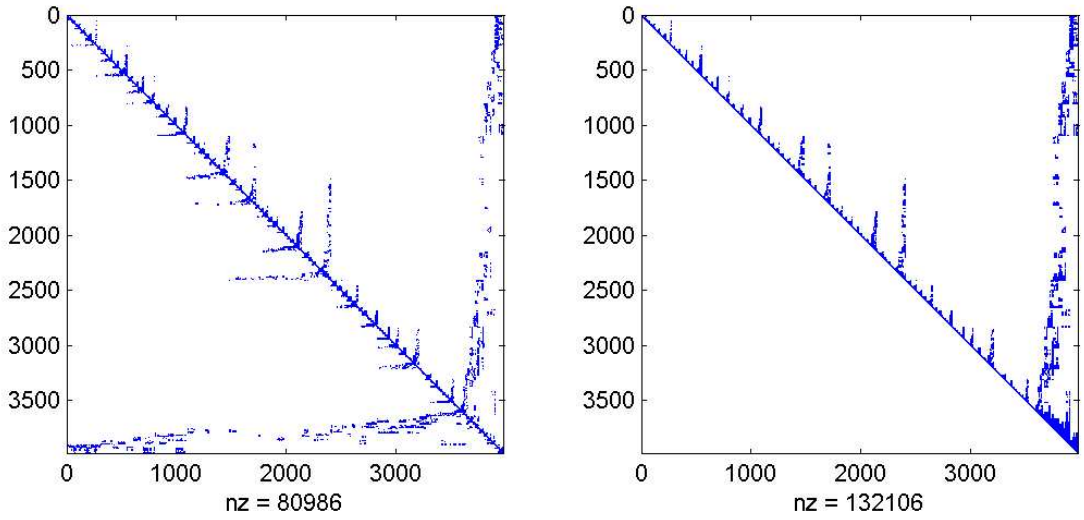


Figure 9: By doing the reordering while taking into account the special block-structure of the SLAM problem, the non-zero count can be eliminated even further, to about 130K, a reduction by a factor 20 with respect to the original R, and substantially less than the 500K entries in the filtering covariance matrix.

- *mfqr*: multifrontal QR factorization [23]
- *qr*: MATLAB built in QR factorization

The results are summarized in Figure 6. We have found that, under those circumstances,

1. The freely available sparse LDL implementation by T. Davis [24] beats MATLAB's built-in Cholesky factorization by about 30%.
2. In MATLAB, the built-in Cholesky beats QR factorization by a large factor.
3. Multifrontal QR factorization is better than MATLAB's QR, but still slower than either Cholesky or LDL.
4. While this is not apparent from the table, using a good column ordering is *much* more important than the choice of factorization algorithm.

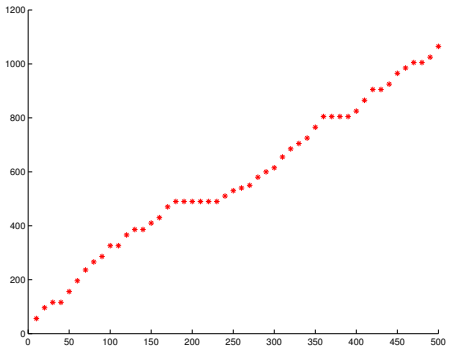
The latter opens up a considerable opportunity for original research in the domain of SLAM, as we found that injecting even a small amount of domain knowledge into that process yields immediate benefits. To illustrate this, we show simulation results for a length 1000 random walk in a 500-landmark environment, corresponding to Figure 5. Both $\mathcal{I}$ and R are shown in Figure 7 for the canonical (and detrimental) ordering with states and landmarks ordered consecutively. The dramatic reduction in fill-in that occurs when using a good re-ordering is illustrated by Figure 8, where we used *colamd* [22]. Finally, when we use the block-oriented ordering heuristic from Section 6, the fill-in drops by another factor of 2.

7.2 Incremental $\sqrt{\text{SAM}}$

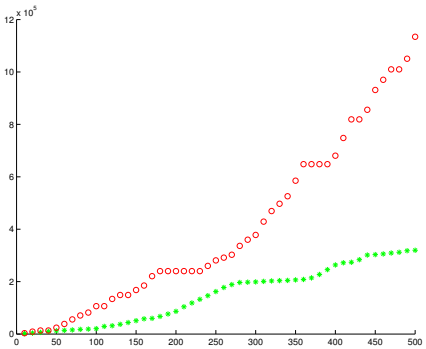
We also compared the performance of an incremental version of $\sqrt{\text{SAM}}$, described in Section 6, with a standard EKF implementation by simulating 500 time steps in a synthetic environment with 2000 landmarks. The results are shown in Figure 10. The factorization of $\mathcal{I}$ was done using sparse LDL [24], while for the column ordering we used *symamd* [22], a version of *colamd* for symmetric positive definite matrices.

Smoothing *every time step* becomes cheaper than the EKF when the number of landmarks N reaches 600. At the end, with $N = 1,100$, each factorization took about 0.6 s., and the slope is nearly linear over time. In contrast, the computational requirements of the EKF increase quadratically with N, and by the end each update of the EKF took over a second.

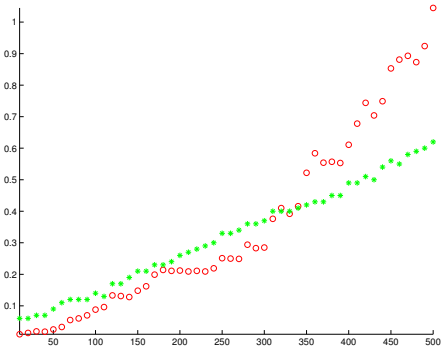
As implementation independent measures, we have also plotted N^2, as well as the number of non-zeros nnz in the Cholesky triangle R. The behavior of the latter is exactly opposite to that of the EKF: when new, unexplored terrain is encountered, there is almost no correlation between new features and the past trajectory and/or map, and nnz stays almost constant.



(a) number of landmarks N seen



(b) N^2 and nnz in R



(c) average time per call, in seconds

Figure 10: Timing results for incremental SAM in a simulated environment with 2000 landmarks, similar to the one in Figure 5, but 10 blocks on the side. As the number of landmarks seen increases, the EKF becomes quadratically slower. Note that the number of non-zeros nnz increases faster when large loops are encountered around $i = 200$ and $i = 350$.

In contrast, the EKF's computation is not affected when re-entering previously visited areas -closing the loop- whereas that is exactly when R fill-in occurs.

We have reason to belief that these results can be further improved: profiling of our implementation showed that roughly 2/3 of the time is spent updating $\mathcal{I}$, which is an artifact of our sparse matrix representation. A compressed row scheme would probably increase the speed by another factor of 2. In addition, incremental QR updating methods should be much faster than doing full factorizations every time.

8 Conclusion

In conclusion, we believe square root information smoothing to be of great practical interest to the SLAM community. It recovers the entire trajectory and is exact, and even the decidedly sub-optimal incremental scheme we evaluated behaves much better than the EKF as the size of the environments grows. In addition, we conjecture that the possibility of re-linearizing the entire trajectory will make $\sqrt{\text{SAM}}$ cope better with noisy measurements governed by non-linear measurement equations. In contrast, non-optimal linearization cannot be recovered from in an EKF, which inevitably *has* to summarize it in a quadratic (Gaussian) approximation.

In this paper we only reported on our initial experiences with this approach, and the following leaves to be desired:

- We have not yet established any tight or amortized complexity bounds that predict the algorithm's excellent performance on problems of a given size.
- The performance results we present are based on simulations. More work is needed to establish that the algorithm performs as well in practice as it does in simulation.
- We have only compared $\sqrt{\text{SAM}}$ against the EKF-based stochastic map algorithm. Comparison of our approach to more recent and faster SLAM variants, both approximate [7, 8, 9] and exact [10], is the object of future work.

Also, we concentrated on the large-scale optimization problem associated with SLAM. Many other issues are crucial in the practice of robot mapping, e.g. the data-association problem. In addition, the techniques exposed here are meant to complement, not replace methods that make judicious approximations in order to reduce the asymptotic complexity of SLAM.

References

[1] R. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *Intl. J. of Robotics Research*, 1987.

[2] J. Leonard, I. Cox, and H. Durrant-Whyte, "Dynamic map building for an autonomous mobile robot," *Intl. J. of Robotics Research*, vol. 11, no. 4, pp. 286–289, 1992.

[3] S. Thrun, "Robotic mapping: a survey," in *Exploring artificial intelligence in the new millennium*. Morgan Kaufmann, Inc., 2003, pp. 1–35.

[4] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in Robotics," in *Autonomous Robot Vehicles*, I. Cox and G. Wilfong, Eds. Springer-Verlag, 1990, pp. 167–193.

[5] J. Castellanos, J. Montiel, J. Neira, and J. Tardos, "The SPmap: A probabilistic framework for simultaneous localization and map building," *IEEE Trans. Robot. Automat.*, vol. 15, no. 5, pp. 948–953, 1999.

[6] M. Dissanayake, P. Newman, H. Durrant-Whyte, S. Clark, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Trans. Robot. Automat.*, vol. 17, no. 3, pp. 229–241, 2001.

[7] J. J. Leonard and H. J. S. Feder, "Decoupled stochastic mapping," *IEEE Journal of Oceanic Engineering*, pp. 561–571, October 2001.

[8] M. Paskin, "Thin junction tree filters for simultaneous localization and mapping," in *Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, 2003.

[9] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *Intl. J. of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.

[10] J. Guivant and E. Nebot, "Optimization of the simultaneous localization and map building algorithm for real time implementation," *IEEE Trans. Robot. Automat.*, vol. 17, no. 3, pp. 242–257, June 2001.

[11] G. Bierman, *Factorization methods for discrete sequential estimation*, ser. Mathematics in Science and Engineering. New York: Academic Press, 1977, vol. 128.

[12] P. Maybeck, *Stochastic Models, Estimation and Control*. New York: Academic Press, 1979, vol. 1.

[13] K. Murphy, "Bayesian map learning in dynamic environments," in *Advances in Neural Information Processing Systems (NIPS)*, 1999.

[14] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *AAAI Nat. Conf. on Artificial Intelligence*, 2002.

[15] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter, *Probabilistic Networks and Expert Systems*, ser. Statistics for Engineering and Information Science. Springer-Verlag, 1999.

[16] J. Dennis and R. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice-Hall, 1983.

[17] R. Smith, M. Self, and P. Cheeseman, "A stochastic map for uncertain spatial relationships," in *Int. Symp on Robotics Research*, 1987.

[18] J. Leonard and H. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," in *IEEE Int. Workshop on Intelligent Robots and Systems*, 1991, pp. 1442–1447.

[19] G. Golub and C. V. Loan, *Matrix Computations*, 3rd ed. Baltimore: Johns Hopkins University Press, 1996.

[20] G. Winkler, *Image analysis, random fields and dynamic Monte Carlo methods*. Springer Verlag, 1995.

[21] J. Yedidia, W. Freeman, and Y.Weiss, "Generalized belief propagation," in *Advances in Neural Information Processing Systems (NIPS)*, 2000, pp. 689–695.

[22] P. R. Amestoy, T. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[23] P. Matstoms, "Sparse QR factorization in MATLAB," *ACM Trans. Math. Softw.*, vol. 20, no. 1, pp. 136–159, 1994.

[24] T. A. Davis, "Algorithm 8xx: a concise sparse Cholesky factorization package," Univ. of Florida, Tech. Rep. TR-04-001, January 2004, submitted to ACM Trans. Math. Software.

[25] T. Davis and W. Hager, "Modifying a sparse Cholesky factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 20, no. 3, pp. 606–627, 1996.